Efficient Parallel Algorithms for Computing All Pair Shortest Paths in Directed Graphs[1]

Yijie Han[2]* *Victor Y. Pan*[3] ** *John H. Reif****

*Department of Computer Science
University of Kentucky
Lexington, KY 40506

**Department of Mathematics and Computer Science
Lehman College, CUNY
Bronx, NY 10468

***Department of Computer Science
Duke University
Durham, NC 27706

Abstract

We present parallel algorithms for computing all pair shortest paths in directed graphs. Our algorithm has time complexity $O(f(n)/p + I(n)\log n)$ on the PRAM using p processors, where $I(n)$ is $\log n$ on the EREW PRAM, $\log\log n$ on the CRCW PRAM, $f(n)$ is $o(n^3)$. On the randomized CRCW PRAM we are able to achieve time complexity $O(n^3/p + \log n)$ using p processors.

Keywords: Analysis of algorithms, design of algorithms, parallel algorithms, graph algorithms, shortest path.

1 Introduction

A number of known algorithms compute the all pair shortest paths in graphs and digraphs with n vertices by using $O(n^3)$ operations [Di][Fl][J]. All these algorithms, however, use at lease $n-1$ recursive steps in the worst case and thus require at least the order of n time in their parallel implementation, even if the number of available processors is not bounded. $O(n)$ time and n^2 processor bounds can indeed be achieved, for instance, in the straightforward parallelization of the algorithm of [Fl]. (Here and hereafter we assume the customary PRAM models of parallel computing [KR].)

[1]Preliminary version of this paper was presented on the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, June, 1992.

[2]Current address: Electronic Data Systems, Inc., 37350 Ecorse Rd., Romulus, MI 48174.

[3]Supported by NSF Grant CCR 90-20690 and PSC CUNY Awards #661340 and #662478.

NC algorithms are also available for this problem. However, they either need $\Omega(n^3 \log n)$ operations or only work for the more narrow class of the input graphs and/or digraphs (which have the edge weights bounded, say, by a constant or have a family of small separators available) [AGM][DNS][DS][GM][L][PK][PR1][PR2][S]. The recent algorithm of [PP1][PP2] uses $O(\log^{2.5} n)$ parallel time and $O(n^3)$ operations in the case of a general graph with n vertices. In this paper we improve the latter time bound to the new record value of $O(I(n) \log n)$, still using $O(n^3)$ operations. Here $I(n)$ is the time complexity of computing the minimum of n elements using n processors. $I(n)$ is $O(\log n)$ under the EREW (Exclusive Read Exclusive Write) PRAM model, $O(\log \log n)$ under the CRCW (Concurrent Read Concurrent Write) PRAM model [V], and a constant under the randomized CRCW PRAM model [R].

Moreover, the total number of operations involved in our $O(I(n) \log n)$ time algorithm can be decreased to $o(n^3)$ on the EREW PRAM and CRCW PRAM, by applying the results in [Fr][T].

To build our algorithms, we incorporated the well known techniques of [Fl] and of the reduction of the shortest path computation to matrix multiplication over the semirings, but added some new nontrivial techniques of studying paths in graphs and digraphs.

We present our algorithm in three stages to illustrate our intuition behind the algorithms. We first give a simple parallel algorithm with time complexity $O(n^3/p + I(n) \log^{1.5} n)$ using p processors. This algorithm takes $O(n^3)$ operations when we use no more than $O(n^3/(I(n) \log^{1.5} n))$ processors. We then show how to speed up this algorithm to achieve time $O(I(n) \log n)$ using $O(n^3/(I(n) \log^{2/3} n))$ processors. By using more sophisticated ideas we show the time complexity $O(n^3/p + I(n) \log n)$ for the all pair shortest path problem. Straightforward applications of the results in [Fr][T] yield time complexity $O(f(n)/p + I(n) \log n)$, where $f(n) = o(n^3)$.

2 Computing All Pair Shortest Paths

We use numbers $0, 1, \ldots, n-1$ to represent input vertices, and a matrix A such that its entry a_{ij} represents the weight of the arc from i to j. We will use the semiring $(A, \min, +)$, so that A^{n-1} represents the shortest distances between all pairs of vertices of the input graph. An arc is an ordered pair of vertices. A path is a finite sequence of vertices. We may assume that there is an arc between every pair of vertices, some of them may have weight ∞. Our algorithms use matrix multiplications to compute shortest paths. The computation can therefore be viewed as contracting each shortest path to a single arc. For example, for an input matrix A the operation $A := AAA$ contracts the length of every shortest path by at least one half. Thus, if our algorithm contracts every path of length $l < n$ to a single arc, then it computes all pair shortest paths correctly. For a path p, we use (p) to denote the cost (the number of steps or the number of iterations of a loop) for contracting p to a single arc. We also use the following special definition.

Definition 1: For a given integer k,
1. $[i]$ denotes a vertex u such that $ik \leq u \leq (i+1)k - 1$. $[i]$, $0 \leq i < n/k$, form a partition of vertices.

2. $[i,j]$ denotes a vertex u such that $ik \leq u \leq (j+1)k-1$, $[i,j]$ is empty if $j < i$.
3. $[i,j][g,h]$ denotes an arc (u,v) such that $ik \leq u \leq (j+1)k-1$ and $gk \leq v \leq (h+1)k-1$.
4. $[i,j]^*$ denotes an empty path or a path $[i,j][i,j]\cdots[i,j]$ of length $< n$.

2.1 A Simple Parallel Algorithm

The input matrix A is divided into submatrices. Each submatrix is a $k \times k$ matrix. For convenience, assume that k divides n, and similarly we assume that the values of all logarithms, powers and ratios below are integers where this is needed. There are a total of n^2/k^2 submatrices A_{ij}, $0 \leq i, j < n/k$. Submatrix A_{ij} contains elements in rows ik to $(i+1)k-1$ and columns jk to $(j+1)k-1$ of A (see Fig. 1). Our next algorithm combines the techniques of Floyd's algorithm and of path computation by means of matrix multiplication.

Algorithm **APSP1**

for $t := 0$ **to** $n/k-1$ **do**
begin
/*Compute the transitive closure of A_{tt} using matrix multiplication. */
$A_{tt} := A_{tt}^*$;
for all i, j, $0 \leq i,j < n/k$, **do** in parallel
$A_{ij} := \min\{A_{ij},\ A_{it}A_{tt}A_{tj}\}$;
end

Let (p) denote the number of iterations of the loop indexed by t in algorithm APSP1 that is needed to contract p to a single arc, where p is a path of length less than n. Fixing k in the definition 1, we have:

Lemma 1: $([0,\ n/k-1][0,\ t]^*[0,\ n/k-1]) \leq t+1$, $0 \leq t < n/k$.

Proof: By induction. Before the iteration $t < 0$, $[0,\ n/k-1][0,\ t]^*[0,\ n/k-1] = [0,\ n/k-1][0,\ n/k-1]$, which is a single arc. Now assuming that the lemma is true for t, we show that it is true for $t+1$.

We consider three cases for a path $p = [0,\ n/k-1][0,\ t+1]^*[0,\ n/k-1]$.

(1) The interior vertices of p do not contain a vertex in $[t+1]$. That is, p is of the form $[0,\ n/k-1][0,\ t]^*[0,\ n/k-1]$. In this case $(p) \leq t+1$, by the induction hypothesis.

(2) The interior vertices of p contain one vertex in $[t+1]$. That is, p is of the form $[0,\ n/k-1][0,\ t]^*[t+1][0,\ t]^*[0,\ n/k-1]$. In this case $p = p_1p_2$, where $p_1 = [0,\ n/k-1][0,\ t]^*[t+1]$, $p_2 = [t+1][0,\ t]^*[0,\ n/k-1]$. By the induction hypothesis, $(p_1) \leq t+1$, $(p_2) \leq t+1$. Therefore, immediately after the t-th iteration (note that t starts at 0), p_1 has been contracted to $[0,\ n/k-1][t+1]$, and p_2 has been contracted to $[t+1][0,\ n/k-1]$. Thus $p = p_1p_2$ has been contracted to $[0,\ n/k-1][t+1][0,\ n/k-1]$. In the $(t+1)$-st iteration, instruction $A_{ij} := \min\{A_{ij},\ A_{i,t+1}A_{t+1,t+1}A_{t+1,j}\}$, $0 \leq i,j < n/k$, contracts the path to a single arc $[0,\ n/k-1][0,\ n/k-1]$. Therefore, $(p) \leq t+2$.

(3) The interior vertices of p contain more than one vertex in $[t+1]$. That is, p is of the form $[0,\ n/k-1][0,\ t]^*[t+1][0,\ t+1]^*[t+1][0,\ t]^*[0,\ n/k-1]$. In this case $p = p_1p_2p_3$, where $p_1 = [0,\ n/k-1][0,\ t]^*[t+1]$, $p_2 = [t+1][0,\ t+1]^*[t+1]$, and $p_3 = [t+1][0,\ t]^*[0,\ n/k-1]$. p_2 can

further be decomposed into blocks, where each block is of the form $[t+1][0,\ t]^*[t+1]$. That is, each block starts at a vertex in $[t+1]$, ends with a vertex in $[t+1]$, but goes through only vertices in $[0,\ t]$. By the induction hypothesis, $(p_1) \leq t+1$, $(p_3) \leq t+1$. Let p' be any block of p_2. We also have that $(p') \leq t+1$, by the induction hypothesis. Therefore, immediately after the t-th iteration, p_1 has been contracted to $[0,\ n/k-1][t+1]$, p_3 has been contracted to $[t+1][0,\ n/k-1]$, and each block of p_2 has been contracted to $[t+1][t+1]$. Thus p_2 has been contracted to $[t+1]^*$. In the $(t+1)$-st iteration, instruction $A_{t+1,t+1} := A^*_{t+1,t+1}$ contracts p_2 to $[t+1][t+1]$. Therefore, p has been contracted to $[0,\ n/k-1][t+1][t+1][0,\ n/k-1]$. Finally, in the $(t+1)$-st iteration, instruction $A_{ij} := \min\{A_{ij},\ A_{i,t+1}A_{t+1,t+1}A_{t+1,j}\}$, $0 \leq i,j < n/k$, contractes p to a single arc $[0,\ n/k-1][0,\ n/k-1]$. Therefore, $(p) \leq t+2$. □

Theorem 2: Algorithm APSP1 correctly computes all pair shortest paths.

Proof: Setting $t = n/k - 1$ in Lemma 1 proves this theorem. □

Theorem 3: The time complexity of algorithm APSP1 is $O((n^3+nk^2\log k)/p+(nI(n)\log n)/k)$.

Proof: Each iteration of the loop indexed by t executes a transitive closure operation for A_{tt}, and n^2/k^2 matrix multiplications and minimizations $A_{ij} := \min\{A_{ij},\ A_{it}A_{tt}A_{tj}\}$. In each iteration the transitive closure requires $O(k^3\log k)$ operations and n^2/k^2 matrix multiplications and minimizations take $O(n^2/k^2 * k^3) = O(n^2k)$ operations. Therefore, the whole algorithm takes $O(n^3 + nk^2\log k)$ operations. Each iteration can be done in time $O(I(n)\log n)$ if enough processors are available. Thus the theorem is proved. □

Setting k to $n/\log^{0.5} n$, we obtain a parallel algorithm with time complexity $O(n^3/p + I(n)\log^{1.5} n)$.

2.2 Speeding Up the Algorithm

So far we were unable to decrease the time complexity below $O(I(n)\log^{1.5} n)$ with $O(n^3)$ operations because we do the transitive closure of A_{tt} sequentially for $t = 0,\ 1,\ \ldots,\ n/k-1$. The loop indexed by t in algorithm APSP1 represents the serialism of the algorithm. We now speed up our algorithm by adopting a new design. In the following algorithm we let $k = n/\log^{1/3} n$ and $x = \log^{2/3} n$.

Algorithm **APSP2**

```
for t := 1 to 5 log n do
  begin
    for all s, 0 ≤ s < n/k, do in parallel
      A_ss := A_ss A_ss A_ss;
    if t mod x = 0 then A := AAA;
  end
```

The intuition behind the design of APSP2 is as follows. We multiply all the diagonal matrices A_{ss}, $0 \leq s < n/k$, simultaneously in each iteration. By doing so we hope to eliminate the serialism in APSP1. We also replace the instruction

```
for all i, j, 0 ≤ i,j < n/k, do in parallel
  A_ij := min{A_ij, A_it A_tt A_tj};
```

in APSP1 with instruction

if $t \bmod x = 0$ **then** $A := AAA$;

APSP2 has $5 \log n$ iterations. Each iteration executes a constant number of matrix multiplications. Therefore, if a sufficient number of processors are available APSP2 can be executed in $O(I(n) \log n)$ time. Also note that the number of operations executed by APSP2 cannot be bounded by $O(n^3)$ because instruction $A := AAA$ is executed more than a constant number of times.

Let (p) be the number of iterations of the loop indexed by t in algorithm APSP2 that is needed to contract p to a single arc. Fixing k in definition 1, we have:

Lemma 4: If $p = [i]^*$ for any $0 \leq i < n/k$, then $(p) \leq \log |p|$, where $|p|$ is the length of p.

Proof: Each iteration reduces the length of such a path by at least one half until the path has been contracted to a single arc. □

In the following we will prove inequalities of the form $(p_1) \leq (p_2) + n$, where p_1 and p_2 are paths and n is a number. In proving such an inequality we always assume that $|p_2| \leq |p_1|$. In fact, in most situations p_2 is a subpath of p_1.

Lemma 5: $([0, n/k-1][0, i]^*[0, n/k-1]) \leq ([0, i]^*) + x$, $1 \leq i \leq n/k$.

Proof: After $[0, i]^*$ has been contracted to a single arc, the instruction $A := AAA$ will be executed within the next x iterations to contract the path to a single arc. □

Lemma 6: If $([i][0, i-1]^*[i]) \leq z(l) + \log l$, where l is the length of the path (*i.e.* there is a term $\log l$ in the expression upbounding $([i][0, i-1]^*[i])$) and $z(l)$ is a nondecreasing function of l, then $([i][0, i]^*[i]) \leq z(l) + \log l + \log\log l + 3$, $1 \leq i < n/k$.

Proof: Let $p = [i][0, i]^*[i]$. Let $|p| = L$. We will partition p into blocks which start from a vertex in $[i]$, end with a vertex in $[i]$, but only go through vertices in $[0, i-1]$. A block must go through at least one vertex in $[0, i-1]$. Thus a block can be denoted by $[i][0, i-1][0, i-1]^*[i]$. p has no more than $L/2^t$ blocks of length $\geq 2^t$. We bound N, the number of arcs of the form $[i][i]$ in p. We note that when a block is contracted to a single arc $[i][i]$ (we say this block is removed or eliminated), it contributes 1 to N.

Example: Let $p = [i][0][2][i][i][i-1][3][4][i][i][i][2][i][3][4][i]$. p has 4 blocks. $N = 3$. Assume that p is contracted to $p_1 = [i][0][i][i][i][i][i][2][i][i]$. Then p_1 has 2 blocks and $N = 5$. Assume now that p_1 is contracted to $p_2 = [i][0][i][i][i]$. Then p_2 has 1 block and $N = 2$.

We note that as a path p is being contracted, the length of p is decreased, the number of blocks is decreased, N may increase. There are two factors affect N. When a block is contracted to a single arc, the block is removed, but it contributes 1 to N. On the other hand, the contraction of p also contracts subpaths of the form $[i]^*$, thus decreasing N.

Let $P = [i][0, i-1]^*[i]$, $|P| = l \leq L$. By the assumption of the lemma we have $(P) \leq z(l) + \log l \leq z(L) + \log l$. All blocks of length $\leq 2^1$ in path p are removed after $z(L)+1$ iterations. There are at most L blocks removed. Therefore after $z(L)+1$ iterations, these blocks contribute at most L to N. All blocks of length $\leq 2^2$ in path p are removed after $z(L)+2$ iterations. There are at most $L/2$ blocks removed. Therefore after $z(L)+2$

iterations, these blocks contribute at most $L/2$ to N. In general, all blocks of length $\leq 2^{t+1}$ in path p are removed after $z(L)+t+1$ iterations. There are at most $L/2^t$ blocks removed. Therefore after $z(L)+t+1$ iterations, these blocks contribute at most $L/2^t$ to N.

Now we count how many arcs $[i][i]$ are removed after each iteration. Because instruction

for all s, $0 \leq s < n/k$, **do** in parallel
$A_{ss} := A_{ss}A_{ss}A_{ss}$;

is executed in each iteration, the length of a subpath $[i]^*$ is cut by at least half after each iteration. Thus it seems that we can reduce N by half in each iteration simply because of this instruction. Such counting is not accurate. Consider a path

$p = [i][2][3][i][i][4][5][i][i][2][0][i][i][3][i-1][i]$.

N is 3 for p. After an iteration (assuming that instruction $A := AAA$ is not executed in this iteration), N is not changed because a single arc $[i][i]$ cannot be removed by the iteration. However, when an arc $[i][i]$ does not contribute to the decreasing of N, it is because the $[i][i]$ is adjacent to a block. After $z(L)+t$ iterations, all blocks of length smaller than 2^t are removed. Therefore, after $z(L)+t$ iterations there are at most $L/2^t$ blocks left, and there are at most $L/2^t$ arcs $[i][i]$ which do not contribute to the decreasing of N.

After $z(L)$ iterations, $N \leq L$. After $z(L)+1$ iterations, we do

$$
\begin{array}{lll}
N & \leq N/2 & /* \text{Because the length of each subpath } [i]^* \text{ is reduced by half } */ \\
 & +L & /* \text{Because there are at most } L \text{ arcs } [i][i] \text{ not contributing to the decreasing of } N. */ \\
 & +L & /* \text{Because there are at most } L \text{ blocks of length } \leq 2^1 \text{ removed.} */
\end{array}
$$

Therefore $N \leq 5L/2$.

After $z(L)+2$ iterations $N \leq N/2 + L/2 + L/2$. Therefore $N \leq 9L/4$. In general, after $z(L)+t$ iterations $N \leq N/2 + L/2^{t-1} + L/2^{t-1}$, we have $N \leq (4t+1)L/2^t$. After $z(L)+\log L+1$ iterations, all blocks are removed, and $N \leq (4(\log L+1)+1)L/2L < 2\log L+3$. At this moment p is contracted to a path of the form $[i]^*$ and of length $\leq N$. It takes at most $\log(2\log L+3) < \log\log L+2$ more iterations to contract the path to a single arc. Thus $(p) \leq z(L)+\log L+\log\log L+3$. □

Lemma 7: $([0,i]^*) \leq \log l + i(3x+\log\log l+3)$, $0 \leq i < n/k$, where l is the length of the path.

Proof: By induction. For $i=0$ we have $([0,0]^*) \leq \log l$ by Lemma 4. Now, assuming that the lemma holds for $i-1$, we show that it holds for i.

A path $p = [0,i]^*$ may take the following forms:

(1). p contains no vertex in $[i]$. p is of the form $[0,i-1]^*$. In this case the lemma is true by the induction hypothesis.

(2). p contains exactly one vertex in $[i]$. p is of the form $[0,i-1]^*[i][0,i-1]^*$. Assume that $[i]$ is neither the starting vertex nor the ending vertex of p (we leave to the reader the case where $[i]$ is the starting or the ending vertex). $p = p_1p_2p_3$, where $p_1 = [0,i-1]^*$, $p_2 = [0,i-1][i][0,i-1]$, $p_3 = [0,i-1]^*$. We assume that both p_1 and p_2 are not empty

and leave to the reader the case when they are empty. After $([0, i-1]^*)$ iterations p_1 is contracted to $[0, i-1][0, i-1]$, p_3 is contracted to $[0, i-1][0, i-1]$. Therefore p is contracted to $[0, i-1][0, i-1][i][0, i-1][0, i-1]$. It takes at most $2x$ more iterations for the instruction $A := AAA$ to be executed twice to contract the whole path to a single arc. Therefore $(p) \leq ([0, i-1]^*) + 2x$. Therefore, the lemma is true.

(3). p contains more than one vertex in $[i]$. p is of the form $[0, i-1]^*[i][0, i]^*[i][0, i-1]^*$. $p = p_1p_2p_3p_4p_5$, where $p_1 = [0, i-1]^*$, $p_2 = [0, i-1][i]$, $p_3 = [i][0, i]^*[i]$, $p_4 = [i][0, i-1]$, $p_5 = [0, i-1]^*$. We assume that p_1, p_2, p_3, p_4, p_5 are not empty and leave to the reader the case where some of them are empty. After $([0, i-1]^*)$ iterations p_1 is contracted to $[0, i-1][0, i-1]$, p_5 is contracted to $[0, i-1][0, i-1]$. Now $([i][0, i-1]^*[i]) \leq ([0, i-1]^*) + x$ (by Lemma 5) $\leq \log l + (i-1)(3x + \log\log l + 3) + x$ (by induction hypothesis) $= z(l) + \log l$. Therefore by Lemma 6, $(p_3) \leq \log l + (i-1)(3x + \log\log l + 3) + x + \log\log l + 3 = T$. Therefore, after T iterations p is contracted to $[0, i-1][0, i-1][i][i][0, i-1][0, i-1]$. It takes at most $2x$ more iterations for the instruction $A := AAA$ to be executed twice to contract the whole path to a single arc. Therefore, $(p) \leq T + 2x = \log l + i(3x + \log\log l + 3)$. □

Theorem 8: Algorithm APSP2 correctly computes all pair shortest paths in time $O(n^3 \log^{1/3} n/p + I(n) \log n)$.

Proof: Setting $i = n/k - 1$ and $l = n$ in Lemma 7 proves that algorithm APSP2 computes all pair shortest paths in $\log n + (n/k - 1)(3x + \log\log n + 3) \leq 5 \log n$ iterations. If enough processors are available each iteration can be executed in $O(I(n))$ time. Instruction $A_{ss} := A_{ss}A_{ss}A_{ss}$ is executed $O(\log n)$ times for all $n/k = \log^{1/3} n$ submatrices, which takes $O(k^3(n/k) \log n) = O(n^3 \log^{1/3} n)$ operations. Instruction $A := AAA$ is executed $O(\log n/x)$ times, which takes $O(n^3 \log^{1/3} n)$ operations. □

If we use APSP2 for the purpose of computing A^*_{tt} in APSP1, we get

Corollary: All pair shortest paths can be computed in time $O(n^3/p + I(n) \log^{7/6} n)$ using p processors.

Remark: Algorithms APSP1 and APSP2 sequentially evaluate the transitive closure of the matrices on the diagonal. This property is useful for reducing the complexity of computing the all pair shortest paths in graphs with a family of precomputed separators[PR1] [PR2].

2.3 A Faster Algorithm

We divide matrix A into levels (see Fig. 2). The 0-th level is $A_0^{(0)} = A$. For each matrix $B = A_i^{(j)}$ at level j we divide it into four submatrices of equal size, $B_{0,0}$, $B_{0,1}$, $B_{1,0}$, $B_{1,1}$, and define $A_{2i}^{(j+1)} = B_{0,0}$ and $A_{2i+1}^{(j+1)} = B_{1,1}$. Thus there is a total of 2^j matrices $A_i^{(j)}$ at level j, each of size $n/2^j \times n/2^j$. We involve the matrices up to level $L = (\log\log n)/2$, and there are $2^L = \log^{1/2} n$ matrices at that level, each of size $n/2^L \times n/2^L$. Let K be the largest number, which is both a power of 3 and less than or equal to $\log n$.

Algorithm **APSP3**

for $t := 1$ **to** $8 \log n$ **do**

begin

for all s, $0 \leq s < 2^L$, **do** in parallel
/* Do matrix multiplication for all matrices at level L.*/
$A_s^{(L)} := A_s^{(L)} A_s^{(L)} A_s^{(L)}$;
if $0 \leq i \leq L-1$ and i is the smallest j such that $t \bmod (K/3^j) = 0$ **then**
/* Do matrix multiplication for all matrices at level i. */
for all s, $0 \leq s < 2^i$, **do** in parallel
$A_s^{(i)} := A_s^{(i)} A_s^{(i)} A_s^{(i)}$;
end

APSP3 executes $A_s^{(L)} := A_s^{(L)} A_s^{(L)} A_s^{(L)}$ in every iteration. For matrices at other levels, they are being multiplied as follows. For the time interval from 1 to $8 \log n$, we let $T_0 = \{t \mid t \bmod K = 0\}$. Instruction $A_s^{(0)} := A_s^{(0)} A_s^{(0)} A_s^{(0)}$ is executed for all $t \in T_0$. Note that because of the way K is defined, T_0 contains only a constant number of elements. Let $T_1 = \{t \mid t \bmod (K/3) = 0\} - T_0$. Instruction $A_s^{(1)} := A_s^{(1)} A_s^{(1)} A_s^{(1)}$ is executed for all $t \in T_1$. The reason T_0 is being subtracted is that $A_s^{(1)}$ is a submatrix of $A_{\lfloor s/2 \rfloor}^{(0)}$, thus operation $A_{\lfloor s/2 \rfloor}^{(0)} := A_{\lfloor s/2 \rfloor}^{(0)} A_{\lfloor s/2 \rfloor}^{(0)} A_{\lfloor s/2 \rfloor}^{(0)}$ "contains" operation $A_s^{(1)} := A_s^{(1)} A_s^{(1)} A_s^{(1)}$. Thus we may view that instruction $A_s^{(1)} := A_s^{(1)} A_s^{(1)} A_s^{(1)}$ is executed every $K/3$ iterations. In general. let $T_i = \{t \mid t \bmod (K/3^i) = 0\} - \sum_{j=0}^{i-1} T_j$, $i < L$. Instruction $A_s^{(i)} := A_s^{(i)} A_s^{(i)} A_s^{(i)}$ is executed for all $t \in T_i$. We may view that instruction $A_s^{(i)} := A_s^{(i)} A_s^{(i)} A_s^{(i)}$ is executed every $K/3^i$ iterations.

Let $k = n/2^L$. By definition 1, $[i]$ denotes any vertex in $\{in/2^L, in/2^L + 1, ..., (i+1)n/2^L - 1\}$. Denote $K/3^{L-i-1}$ by t_i, $0 \leq i \leq L-1$. Then instruction $A_s^{(L-i-1)} := A_s^{(L-i-1)} A_s^{(L-i-1)} A_s^{(L-i-1)}$ can be viewed as being executed every t_i iterations, and instruction $A_s^{(L-1)} := A_s^{(L-1)} A_s^{(L-1)} A_s^{(L-1)}$ can be viewed as being executed every t_0 iterations.

To analyze paths being contracted by APSP3, we define function $cost(x, l)$ as follows.

$cost(0, l) = \log l$;
$cost(1, l) = \log l + 2t_0 + \log\log l + 3$;
If $x \neq 0$ and x is even, $cost(x, l) = cost(x-1, l) + t_i$, where i is the largest integer j such that $x/2^j$ is odd;
If $x \neq 1$ and x is odd, $cost(x, l) = cost(x-1, l) + t_0 + 2(\log\log l + 3) + \sum_{j=0}^{i-1} t_j$, where i is the largest integer j such that $(x+1)/2^j$ is odd.

Let p be a path, let (p) be the number of iterations needed in APSP3 to contract p to a single arc. Let l be the maximum length of the path under consideration. We define function $COST(x, l)$ as follows.

$COST(0, l) = ([0]^*)$;
$COST(1, l) = ([0, 1]^*)$;
If $x \neq 0$ and x is even, $COST(x, l) = ([x, x + 2^i - 1][0, x-1]^*[x, x + 2^i - 1])$, where i is the largest integer j such that $x/2^j$ is odd;
If $x \neq 1$ and x is odd, $COST(x, l) = ([x - 2^i + 1, x][0, x]^*[x - 2^i + 1, x])$, where i is the largest integer j such that $(x+1)/2^j$ is odd.

The definition of $COST$ is for paths of length $\leq l$. For example, $COST(0, l)$ is obtained

by taking the maximum of the number of iterations needed to contract a path over the paths of the form $[0]^*$ and of length $\leq l$.

We give a simple way to figure out the formula we are evaluating.

For even x, $x \neq 0$, we define $COST(x, l) = ([a, b][0, x-1]^*[a, b])$, where a and b can be found out in Fig. 2. We first find the largest square in which x is at the top left corner. For $x = 2$ this square is the square containing 2, 3. For $x = 4$ this square is the square containing 4,5,6,7. For $x = 6$ this square is the square containing 6,7. a is the smallest number in the square, that is, the number at the top left of the square. b is the largest number in the square, that is, the number at the bottom right of the square. For even x, $x \neq 0$, we also define $cost(x, l) = cost(x-1, l) + t_i$, where t_i can be found out in Fig. 2. We first find the largest square in which x is at the top left corner. Then t_i is the label in the immediate left neighbor square. Note that for all even x, $x \neq 0$, the squares containing t_i's do not overlap. If we sum all t_i for all even x, $x \neq 0$, we are in fact summing all t_i's except t_0's in Fig. 2.

For odd x, $x \neq 1$, we define $COST(x, l) = ([a, b][0, x]^*[a, b])$, where a and b can be found out in Fig. 2. We first find the largest square in which x is at the bottom right corner. For $x = 3$ this square is the square containing 0,1,2,3. For $x = 5$ this square is the square containing 4,5. For $x = 7$ this square is the square containing 0 to 7. a is the smallest number in the square. b is the largest number in the square. For odd x, $x \neq 1$, we also define $cost(x, l) = cost(x-1, l) + t_0 + 2(\log\log l + 3) + \sum t_j$, where $\sum t_j$ can be found out in Fig. 2. We first find the largest square in which x is at the bottom right corner. The t_j's in $\sum t_j$ are the t_j's in the bottom "row" of the square. Note that for all odd x, $x \neq 0$, the squares containing $\sum t_j$'s do not overlap. If we sum all $\sum t_j$ for all odd x, $x \neq 0$, we are in fact summing all t_i's in Fig. 2 except the t_0 in the top left square containing 0 and 1. Now consider the extra t_0. Considering the fact that x_0's are not present in the formula for $cost(x, l)$, x is even and $x \neq 0$, if we sum all t_i for all x except $x = 0, 1$, we are in fact summing all t_i in Fig. 2 twice except the t_0 in the top left square containing 0 and 1. However, t_0 is counted twice in the formula for $cost(1, l)$. Therefore, if we sum t_i for all x, we are in fact summing all t_i in Fig. 2 exactly twice.

We use several properties of APSP3 in the analysis of paths.

One property we use is symmetry. We note that vertices in $[2i]$ and vertices in $[2i+1]$ are symmetrical with respect to APSP3. For example, path $p_1 = [1][0, 1]^*[1]$ is symmetrical to path $p_2 = [0][0, 1]^*[0]$. That is to say, if we replace $[1]$ in p_1 with $[0]$ and replace $[0]$ with $[1]$ we obtain p_2 as the symmetrical path of p_1. Because a matrix multiplication at level $\leq L-1$ in APSP3 either involves both $[0]$ and $[1]$ or involves neither $[0]$ nor $[1]$, and matrix multiplications at level L involves $[0]$ and $[1]$. Therefore, the way that p_1 and p_2 are contracted is the same as far as APSP3 is used to contract p_1 and p_2. Therefore, $(p_1) = (p_2)$. We also have that $p_1 = [2][1]^*[2]$ is symmetrical to $p_2 = [2][0]^*[2]$, and $p_3 = [4, 5][0, 7]^*[4, 5]$ is symmetrical to $p_4 = [6, 7][0, 7]^*[6, 7]$ (refer to Fig. 2). We also use symmetry in a somewhat generalized sense. For example, $p_5 = [6][0, 5]^*[7]$ is symmetrical to $p_6 = [6][0, 5]^*[6]$ because matrix multiplications for contracting p_5 and p_6 proceeds in the same pattern. Also, $p_7 = [6, 7][0, 3]^*[4, 5]$ is symmetrical to $p_8 = [6, 7][0, 3]^*[6, 7]$. We have $(p_1) = (p_2)$ if p_1 and p_2 are symmetrical.

The second property we will use is reverse. The reversed path of $p_1 = [1][0][2][3]$ is $p_2 = [3][2][0][1]$. That is, the reversed path is obtained by reversing the vertices of the original path. It is not difficult to see that if p_1 is the reversed path of p_2 then $(p_1) = (p_2)$.

The third property we will use is proper subset. If the set of paths represented by p_1 is contained in the set of paths represented by path p_2 then we say that p_1 is a subset of p_2. If the containment is proper then we say p_1 is a proper subset of p_2. For example, $[0, 1][5, 6]^*[0, 1]$ is a proper subset of $[0, 3][5, 6]^*[0, 3]$. If p_1 is a proper subset of p_2 then $(p_1) \leq (p_2)$.

We use four lemmas below to prove $COST(x, l) \leq cost(x, l)$. We use an implied induction. That is, when we are proving $COST(x, l) \leq cost(x, l)$ we assume that $COST(y, l) \leq cost(y, l)$ is true for all $0 \leq y < x$.

To help understand the following proofs, we suggest that the reader try several particular values for x and refer to Fig. 2 for each case analyzed.

Lemma 9: $COST(0, l) \leq cost(0, l)$.

Proof: Each iteration contracts such a path by at least half. Therefore, the Lemma is true. □

Lemma 10: $COST(1, l) \leq cost(1, l)$.

Proof: We have the following cases for a path $p = [0, 1]^*$:

(a1). p is of the forms $[0]^*$ or $[1]^*$, then $(p) \leq \log l$ because instruction $A_s^{(L)} := A_s^{(L)} A_s^{(L)} A_s^{(L)}$ is executed in each iteration. .

(a2). p is of the form $[1][0]^*[1]$. After $[0]^*$ is contracted to a single arc, it takes at most t_0 iterations for the instruction $A_s^{(L-1)} := A_s^{(L-1)} A_s^{(L-1)} A_s^{(L-1)}$ to be executed in order to contract the path to a single arc. Therefore, $(p) \leq ([0]^*) + t_0 \leq \log l + t_0$ (by (a1)).

(a3). p is of the form $[1][0, 1]^*[1]$, then $(p) \leq \log l + t_0 + \log\log l + 3$ (by (a2) and Lemma 6).

(a4). p is of the form $[0][0, 1]^*[0]$. This case is a symmetrical case of (a3). By symmetry we have $(p) \leq \log l + t_0 + \log\log l + 3$.

(a5). p is of the form $[0][0, 1]^*[1]$. Assume that p has at least two vertices in $[0]$ (we leave to the reader the case that p has only one vertex in $[0]$). p can be written as $p_1 p_2 p_3$, where $p_1 = [0][0, 1]^*[0]$, $p_2 = [0][1]$, $p_3 = [1]^*$. By (a4), $(p_1) \leq \log l + t_0 + \log\log l + 3$. By (a1), $(p_3) \leq \log l$. Thus after $\log l + t_0 + \log\log l + 3$ iterations, p_1 is contracted to a single arc $[0][0]$, p_3 is contracted to a single arc $[1][1]$ (it is possible p_3 is empty). Therefore p is contracted to $[0][0][1][1]$. Within t_0 iterations instruction $A_s^{(L-1)} := A_s^{(L-1)} A_s^{(L-1)} A_s^{(L-1)}$ will be executed, which contracts p to a single arc. Therefore $(p) \leq \log l + 2t_0 + \log\log l + 3$.

(a6). p is of the form $[1][0, 1]^*[0]$. This case is symmetrical to (a5).

Now a path $p = [0, 1]^*$ can be in only one of the following two situations.

(1). p only contains vertices in $[0]$, or p only contains vertices in $[1]$. This situation is taken care of in (a1).

(2). p contains vertices both in $[0]$ and in $[1]$. We look at the starting and ending vertices of p. If the starting ending vertex pair $< a, b >$ is $< [1], [1] >$, the situation is calculated

in (a3). If the pair is $< [0], [0] >$, it is calculated in (a4). If the pair is $< [0], [1] >$, it is calculated in (a5). Finally, if the pair is $< [1], [0] >$, it is calculated in (a6). $\Box$

Lemma 11: For even x, $x \neq 0$, $COST(x, l) = ([x, x + 2^i - 1][0, x - 1]^*[x, x + 2^i - 1]) \leq cost(x, l)$, where i is the largest integer j such that $x/2^j$ is odd.

Proof: For $x \neq 0$ and x even, and for $p = [x, x + 2^i - 1][0, x - 1]^*[x, x + 2^i - 1]$ we have the following cases.

(b1). x is a power of 2. By the definition of i, $x - 2^i = 0$. When $[0, x - 1]^*$ is contracted to a single arc, p is contracted to $[x, x + 2^i - 1][0, x - 1][0, x - 1][x, x + 2^i - 1]$. Vertices in $[0, x - 1]$ and $[x, x + 2^i - 1]$ are in $A_0^{(L-i-1)}$, therefore it takes at most t_i more iterations to multiply the matrices at level $L - i - 1$, in order to contract the path to a single arc. Therefore $COST(x, l) = ([x, x + 2^i - 1][0, x - 1]^*[x, x + 2^i - 1]) \leq ([0, x - 1]^*) + t_i = COST(x - 1, l) + t_i \leq cost(x - 1, l) + t_i = cost(x, l)$.

(b2). x is not a power of 2. There are three subcases.

(b2.1). p does not have any vertex in $[x - 2^i, x - 1]$. Thus p is of the form $[x, x + 2^i - 1][0, x - 2^i - 1]^*[x, x + 2^i - 1]$. We have $(p) = ([x, x + 2^i - 1][0, x - 2^i - 1]^*[x, x + 2^i - 1]) \leq ([x - 2^i, x + 2^i - 1][0, x - 2^i - 1]^*[x - 2^i, x + 2^i - 1])$ (by proper subset) $= COST(x - 2^i, l) \leq cost(x - 2^i, l) \leq cost(x, l)$.

(b2.2). p contains one vertex in $[x - 2^i, x - 1]$. p is of the form $[x, x + 2^i - 1][0, x - 2^i - 1]^*[x - 2^i, x - 1][0, x - 2^i - 1]^*[x, x + 2^i - 1]$. $p = p_1p_2$, where $p_1 = [x, x + 2^i - 1][0, x - 2^i - 1]^*[x - 2^i, x - 1]$, $p_2 = [x - 2^i, x - 1][0, x - 2^i - 1]^*[x, x + 2^i - 1]$. $(p_1) = ([x, x+2^i-1][0, x-2^i-1]^*[x-2^i, x-1]) \leq ([x-2^i, x+2^i-1][0, x-2^i-1]^*[x-2^i, x+2^i-1])$ (by proper subset) $= COST(x - 2^i, l) \leq cost(x - 2^i, l) \leq cost(x - 1, l)$. After (p_1) iterations p_1 is contracted to a single arc. Because p_2 is the reverse of p_1, p_2 is also contracted to a single arc after (p_1) iterations. Therefore, after (p_1) iterations p is contracted to $p_3 = [x, x+2^i-1][x-2^i, x-1][x, x+2^i-1]$. Vertices in p_3 are all in a submatrix $A_s^{(L-i-1)}$. Therefore, it takes at most t_i more iterations with an execution of matrix multiplication at level $L - i - 1$ to contract p to a single arc. Thus, $(p) \leq cost(x - 1, l) + t_i = cost(x, l)$.

(b2.3). p contains more than one vertex in $[x-2^i, x-1]$. p is of the form $[x, x+2^i-1][0, x-2^i-1]^*[x-2^i, x-1][0, x-1]^*[x-2^i, x-1][0, x-2^i-1]^*[x, x+2^i-1]$. $p = p_1p_2p_3$, where $p_1 = [x, x + 2^i - 1][0, x - 2^i - 1]^*[x - 2^i, x - 1]$, $p_2 = [x - 2^i, x - 1][0, x - 1]^*[x - 2^i, x - 1]$, $p_3 = [x - 2^i, x - 1][0, x - 2^i - 1]^*[x, x + 2^i - 1]$. We have $(p_1) = ([x, x + 2^i - 1][0, x - 2^i - 1]^*[x - 2^i, x - 1]) \leq ([x - 2^i, x + 2^i - 1][0, x - 2^i - 1]^*[x - 2^i, x + 2^i - 1])$ (by proper subset) $= COST(x - 2^i, l) \leq cost(x - 2^i, l) \leq cost(x - 1, l)$. We also have $(p_3) = (p_1)$ (by reverse), and $(p_2) = COST(x - 1, l) \leq cost(x - 1, l)$. Therefore, after $cost(x - 1, l)$ iterations p is contracted to $p_4 = [x, x + 2^i - 1][x - 2^i, x - 1][x - 2^i, x - 1][x, x + 2^i - 1]$. Vertices in p_4 are all in a submatrix $A_s^{(L-i-1)}$. Therefore, it takes at most t_i more iterations with an execution of matrix multiplication at level $L - i - 1$ to contract p to a single arc. Thus, $(p) \leq cost(x - 1, l) + t_i = cost(x, l)$. $\Box$

Lemma 12: For odd x, $x \neq 1$, $COST(x, l) = ([x-2^i+1, x][0, x]^*[x-2^i+1, x]) \leq cost(x, l)$, where i is the largest integer j such that $(x + 1)/2^j$ is odd.

Proof: Let i be the largest integer j such that $(x + 1)/2^j$ is odd, Let $y = x - 1$. Let i' be the largest integer j such that $y/2^j$ is odd. We have the following cases:

(c1). $([x][0, y-1]^*[x]) \leq ([y, y+2^{i'}-1][0, y-1]^*[y, y+2^{i'}-1]) = COST(x-1, l) \leq cost(x-1, l)$. $([x][0, y-1]^*[y]) \leq ([y, y+2^{i'}-1][0, y-1]^*[y, y+2^{i'}-1]) = COST(x-1, l) \leq cost(x-1, l)$. $([y][0, y-1]^*[x]) \leq ([y, y+2^{i'}-1][0, y-1]^*[y, y+2^{i'}-1]) = COST(x-1, l) \leq cost(x-1, l)$. $([y][0, y-1]^*[y]) \leq ([y, y+2^{i'}-1][0, y-1]^*[y, y+2^{i'}-1]) = COST(x-1, l) \leq cost(x-1, l)$. All by proper subset.

(c2). $([y][0, y]^*[y]) \leq cost(x-1, l) + \log\log l + 3$ (by Lemma 6 and (c1), because $cost(x-1, l) = z(l) + \log l$).

(c3). $[x][0, y]^*[x]$ can have three subcases.

(c3.1). $[x][0, y]^*[x]$ does not contain any vertex in $[y]$. Thus it is of the form $[x][0, y-1]^*[x]$. This reduces to (c1).

(c3.2). $[x][0, y]^*[x]$ contains one vertex in $[y]$. It is of the form $p = [x][0, y-1]^*[y][0, y-1]^*[x]$. $p = p_1p_2$, where $p_1 = [x][0, y-1]^*[y]$ and $p_2 = [y][0, y-1]^*[x]$. We have $(p_1) = (p_2)$ (by reverse). After (p_1) iterations p is contracted to $[x][y][x]$. Because $[x]$ and $[y]$ are in a submatrix $A_s^{(L-1)}$, it takes t_0 more iterations to contract p to a single arc. Therefore $([x][0, y]^*[x]) \leq ([x][0, y-1]^*[y]) + t_0$. This reduces to case (c1).

(c3.3). $[x][0, y]^*[x]$ contains more than one vertex in $[y]$. It is of the form $p = [x][0, y-1]^*[y][0, y]^*[y][0, y-1]^*[x]$. $p = p_1p_2p_3$, where $p_1 = [x][0, y-1]^*[y]$, $p_2 = [y][0, y]^*[y]$, $p_3 = [y][0, y-1]^*[x]$. We have $(p_1) = (p_3)$ (by reverse). After $\max\{(p_1), (p_2)\}$ iterations p is contracted to $[x][y][y][x]$. Because $[x]$ and $[y]$ are in a submatrix $A_s^{(L-1)}$, it takes at most t_0 more iterations to contract p to a single arc. Thus $(p) \leq \max\{(p_1), (p_2)\} + t_0$. Now (p_1) is evaluated in case (c1) and (p_2) is evaluated in case (c2).

Summarizing three subcases for (c3) and using (c1) and (c2), we obtain that $([x][0, y]^*[x]) \leq cost(x-1, l) + t_0 + \log\log l + 3$.

(c4). $([x][0, x]^*[x]) \leq cost(x-1, l) + t_0 + \log\log l + 3 + \log\log l + 3$ (by Lemma 6 and (c3)).

(c5). We use i steps to evaluate $([x-2^i+1, x][0, x]^*[x-2^i+1, x])$. We define $cost(x, l, 0) = cost(x-1, l) + t_0 + 2(\log\log l + 3)$ and $cost(x, l, j) = cost(x, l, j-1) + t_{j-1}$, $1 \leq j \leq i$. In step $1 \leq j \leq i$ we evaluate $([x-2^j+1, x][0, x]^*[x-2^j+1, x])$. We show that $([x-2^j+1, x][0, x]^*[x-2^j+1, x]) \leq cost(x, l, j)$. We denote $([x-2^j+1, x][0, x]^*[x-2^j+1, x])$ by $COST(x, l, j)$. Therefore we need establish $COST(x, l, j) \leq cost(x, l, j)$, $1 \leq j \leq i$. We have the following steps:

(c5.1). $([x-2^{j-1}+1, x][0, x]^*[x-2^{j-1}+1, x]) = ([x-2^j+1, x-2^{j-1}][0, x]^*[x-2^j+1, x-2^{j-1}])$, by symmetry.

(c5.2). Evaluate $([x-2^j+1, x-2^{j-1}][0, x]^*[x-2^{j-1}+1, x])$. For such a path p we have the following cases:

(c5.2.1). p contains only one vertex in $[x-2^j+1, x-2^{j-1}]$ and one vertex in $[x-2^{j-1}+1, x]$. Then p is of the form $[x-2^j+1, x-2^{j-1}][0, x-2^j]^*[x-2^{j-1}+1, x]$. We have $(p) = ([x-2^j+1, x-2^{j-1}][0, x-2^j]^*[x-2^{j-1}+1, x]) \leq ([x-2^j+1, x][0, x-2^j]^*[x-2^j+1, x])$ (by proper subset) $= COST(x-2^j+1, l) \leq cost(x-2^j+1, l) \leq cost(x, l, j)$.

(c5.2.2). p contains more than one vertex in $[x-2^j+1, x-2^{j-1}]$ but only one vertex in $[x-2^{j-1}+1, x]$. Let $p = p_1p_2$, where p_1 contains all vertices $[x-2^j+1, x-2^{j-1}]$ in p and ends at the last vertex $[x-2^j+1, x-2^{j-1}]$ in p. p_1 and p_2 can be written as $p_1 = [x-2^j+1, x-$

$2^{j-1}][0,\ x-2^{j-1}]^*[x-2^j+1,\ x-2^{j-1}]$, $p_2 = [x-2^j+1,\ x-2^{j-1}][0,\ x-2^j]^*[x-2^{j-1}+1,\ x]$. $(p_1) = ([x-2^j+1,\ x-2^{j-1}][0,\ x-2^{j-1}]^*[x-2^j+1,\ x-2^{j-1}]) \leq ([x-2^j+1,\ x-2^{j-1}][0,\ x]^*[x-2^j+1,\ x-2^{j-1}])$ (by proper subset) $= ([x-2^{j-1}+1,\ x][0,\ x]^*[x-2^{j-1}+1,\ x])$ (by symmetry, see case (c5.1)) $= COST(x,\ l,\ j-1) \leq cost(x,\ l,\ j-1)$. $(p_2) = ([x-2^j+1,\ x-2^{j-1}][0,\ x-2^j]^*[x-2^{j-1}+1,\ x]) \leq ([x-2^j+1,\ x][0,\ x-2^j]^*[x-2^j+1,\ x])$ (by proper subset) $= COST(x-2^j+1,\ l) \leq cost(x-2^j+1,\ l) \leq cost(x,\ l,\ j-1)$. Therefore, after $cost(x,\ l,\ j-1)$ iterations p_1 is contracted to $[x-2^j+1,\ x-2^{j-1}][x-2^j+1,\ x-2^{j-1}]$, p_2 is contracted to $[x-2^j+1,\ x-2^{j-1}][x-2^{j-1}+1,\ x]$. Therefore p is contracted to $[x-2^j+1,\ x-2^{j-1}][x-2^j+1,\ x-2^{j-1}][x-2^{j-1}+1,\ x]$. It takes at most t_{j-1} more iterations for the instruction $A_s^{(L-j)} := A_s^{(L-j)}A_s^{(L-j)}A_s^{(L-j)}$ to be executed in order to contract p to a single arc. Therefore, $(p) \leq cost(x,\ l,\ j-1) + t_{j-1} = cost(x,\ l,\ j)$.

(c5.2.3). p contains more than one vertex in $[x-2^{j-1}+1,\ x]$ but only one vertex in $[x-2^j+1,\ x-2^{j-1}]$. This case is symmetrical to case (c5.2.2). By symmetry we have $(p) \leq cost(x,\ l,\ j)$.

(c5.2.4). p contains more than one vertex in $[x-2^j+1,\ x-2^{j-1}]$ and more than one vertex in $[x-2^{j-1}+1,\ x]$. Let $p = p_1p_2p_3$, where p_1 contains all vertices $[x-2^j+1,\ x-2^{j-1}]$ in p and ends at the last vertex $[x-2^j+1,\ x-2^{j-1}]$ in p. p_1, p_2 and p_3 can be written as $p_1 = [x-2^j+1,\ x-2^{j-1}][0,\ x]^*[x-2^j+1,\ x-2^{j-1}]$, $p_2 = [x-2^j+1,\ x-2^{j-1}][0,\ x-2^j]^*[x-2^{j-1}+1,\ x]$, $p_3 = [x-2^{j-1}+1,\ x][0,\ x]^*[x-2^{j-1}+1,\ x]$. $(p_1) = ([x-2^j+1,\ x-2^{j-1}][0,\ x]^*[x-2^j+1,\ x-2^{j-1}]) = ([x-2^{j-1}+1,\ x][0,\ x]^*[x-2^{j-1}+1,\ x])$ (by symmetry, see case (c5.1)) $= COST(x,\ l,\ j-1) \leq cost(x,\ l,\ j-1)$. $(p_2) = ([x-2^j+1,\ x-2^{j-1}][0,\ x-2^j]^*[x-2^{j-1}+1,\ x]) \leq ([x-2^j+1,\ x][0,\ x-2^j]^*[x-2^j+1,\ x])$ (by proper subset) $= COST(x-2^j+1,\ l) \leq cost(x-2^j+1,\ l) \leq cost(x,\ l,\ j-1)$. $(p_3) = ([x-2^{j-1}+1,\ x][0,\ x]^*[x-2^{j-1}+1,\ x]) = COST(x,\ l,\ j-1) \leq cost(x,\ l,\ j-1)$. Therefore, after $cost(x,\ l,\ j-1)$ iterations p_1 is contracted to $[x-2^j+1,\ x-2^{j-1}][x-2^j+1,\ x-2^{j-1}]$, p_2 is contracted to $[x-2^j+1,\ x-2^{j-1}][x-2^{j-1}+1,\ x]$, p_3 is contracted to $[x-2^{j-1}+1,\ x][x-2^{j-1}+1,\ x]$. Thus p is contracted to $[x-2^j+1,\ x-2^{j-1}][x-2^j+1,\ x-2^{j-1}][x-2^{j-1}+1,\ x][x-2^{j-1}+1,\ x]$. It takes at most t_{j-1} more iterations for the instruction $A_s^{(L-j)} := A_s^{(L-j)}A_s^{(L-j)}A_s^{(L-j)}$ to be executed in order to contract p to a single arc. Therefore, $(p) \leq cost(x,\ l,\ j-1) + t_{j-1} = cost(x,\ l,\ j)$.

Summarizing (c5.2.1), (c5.2.2), (c5.2.3) and (c5.2.4) we have $(p) = ([x-2^j+1,\ x-2^{j-1}][0,\ x]^*[x-2^{j-1}+1,\ x]) \leq cost(x,\ l,\ j)$.

(c5.3). Evaluating $([x-2^{j-1}+1,\ x][0,\ x]^*[x-2^j+1,\ x-2^{j-1}])$. This is the reversed path of the path analyzed in case (c5.2). Therefore $([x-2^{j-1}+1,\ x][0,\ x]^*[x-2^j+1,\ x-2^{j-1}]) \leq cost(x,\ l,\ j)$.

(c5.4). Now evaluating $([x-2^j+1,\ x][0,\ x]^*[x-2^j+1,\ x]) = COST(x,\ l,\ j)$. The starting ending vertex pair $< a,\ b >$ of path $p = [x-2^j+1,\ x][0,\ x]^*[x-2^j+1,\ x]$ is either $< [x-2^j+1,\ x-2^{j-1}],\ [x-2^j+1,\ x-2^{j-1}] >$ or $< [x-2^{j-1}+1,\ x],\ [x-2^{j-1}+1,\ x] >$ or $< [x-2^j+1,\ x-2^{j-1}],\ [x-2^{j-1}+1,\ x] >$ or $< [x-2^j+1,\ x-2^{j-1}] >$. Therefore p is one of the four paths p_1, p_2, p_3, p_4, where $p_1 = [x-2^j+1,\ x-2^{j-1}][0,\ x]^*[x-2^j+1,\ x-2^{j-1}]$, $p_2 = [x-2^{j-1}+1,\ x][0,\ x]^*[x-2^{j-1}+1,\ x]$, $p_3 = [x-2^j+1,\ x-2^{j-1}][0,\ x]^*[x-2^{j-1}+1,\ x]$, and $p_4 = [x-2^{j-1}+1,\ x][0,\ x]^*[x-2^j+1,\ x-2^{j-1}]$. We have $(p_2) = COST(x,\ l,\ j-1) \leq cost(x,\ l,\ j-1) \leq cost(x,\ l,\ j)$, $(p_1) = (p_2)$ (by symmetry), $(p_3) \leq cost(x,\ l,\ j)$ (by (c5.2)), $(p_4) \leq cost(x,\ l,\ j)$ (by (c5.3)). Therefore we have $(p) \leq cost(x,\ l,\ j)$.

Summarizing (c1) to (c5) we have $COST(x, l) = ([x - 2^i + 1, x][0, x]^*[x - 2^i + 1, x])$ $\leq cost(x, l, i) = cost(x, l, 0) + \sum_{j=0}^{i-1} t_j = cost(x - 1, l) + t_0 + 2(\log\log n + 3) + \sum_{j=0}^{i-1} t_j = cost(x, l)$. □

Theorem 13: $COST(x, l) \leq cost(x, l)$, $0 \leq x < 2^L$.

Proof: By Lemmas 9 to 12. □

Theorem 14: The total number of iterations needed to compute all pair shortest paths in APSP3 is no more than $8 \log n$.

Proof: By Theorem 13, $cost(2^L - 1, n)$ upper bounds the number of iterations needed to contract any path of length less than n to a single arc. We have $cost(2^L - 1, n) \leq \log n + 2^L(\log\log n + 3) + \sum_{j=1}^{L} t_{j-1} 2^{L-j+1} = \log n + 2^L(\log\log n + 3) + \sum_{j=1}^{L} K 2^{L-j+1}/3^{L-j} \leq 2\log n + 6\log n = 8\log n$. As we have explained before, when we sum all t_i's for all x we are in fact summing all t_i's in Fig. 2 twice. We thus get the term $\sum_{j=1}^{L} t_{j-1} 2^{L-j+1}$. $2(\log\log n + 3)$ is included once for each odd x except $x = 1$ where the term is only $\log\log n + 3$. Thus we get the term $2^L(\log\log n + 3)$. The term $\log n$ comes from $cost(1, n)$. □

Theorem 15: Algorithm APSP3 computes all pair shortest paths in time $O(n^3/p + I(n)\log n)$.

Proof: Each iteration can be executed in $I(n)$ time if enough processors are available. Thus we only need to show that the total number of operations executed is $O(n^3)$. Matrices at level i, $0 \leq i < L$, are multiplied with themselves $8\log n / t_{L-i-1} \leq 8\log n 3^i/K \leq 24 * 3^i$ times. There are 2^i such matrices and each matrix multiplication takes $O((n/2^i)^3)$ operations. Thus the total number of operations for these matrix multiplications is $O(\sum_{i=0}^{L-1} 3^i * 2^i * (n/2^i)^3) = O(n^3)$. There are 2^L submatrices at level L. Each of them is multiplied to itself $8\log n$ times. The total number of operations for these matrix multiplications is $O(2^L(n/2^L)^3 * \log n) = O(n^3)$. □

If we multiply two matrices by using the results in [Fr][T], we can reduce the number of operations to $o(n^3)$.

Corollary: Algorithm APSP3 computes all pair shortest paths in time $O(f(n)/p + I(n)\log n)$ on the EREW and CRCW PRAM, where $f(n) = o(n^3)$. □

This corollary does not apply to the randomized CRCW PRAM. Because we multiply two matrices in constant time using n^3 processors on the randomized PRAM, Fredman and Takaoka's technique cannot be applied in constant time to result in savings.

References

[AGM] N. Alon, Z. Galil, O. Margalit. On the exponent of all pairs shortest path problem. *Proc. 32nd Ann. IEEE Symp. FOCS*, 569-575, 1991.

[DNS] E. Dekel, D. Nassimi and S. Sahni. Parallel matrix and graph algorithms. *SIAM J. Comput.* 10, 657-675, 1981.

[DS] E. Dekel and S. Sahni. Binary trees and parallel scheduling algorithms. *IEEE Trans. Comput.* 32, 307-315, 1983.

[Di] E. W. Dijkstra. A note on two problems in connextion with graphs. *Numerische Mathematik* 1, 269-271, 1959.

[Fl] R. N. Floyd. Algorithm 97, Shortest path. *Comm. ACM* 5, 345, 1962.

[Fr] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, Vol. 5, No. 1, 83-89(March 1976).

[GM] H. Gazit and G. L. Miller. An improved parallel algorithm that computes the BFS numbering of a directed graph. *Information Processing Letters* 28:1, 61-65, 1988.

[J] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24, 1, 1-13, 1977.

[KR] M. Karp, V. Ramachandran. A survey of parallel algorithms for shared memory machines. "*Handbook of Theoretical Computer Science,*" 869-941, North-Holland, Amsterdam, 1990.

[L] A. Lingas. Efficient parallel algorithms for path problems in planar directed graphs. *Proc. SIGAL'90, Lecture Notes in Computer Science* 450, 447-457.

[PK] R. C. Paige and C. P. Kruskal. Parallel algorithms for shortest path problems. *Proc. 1985 Int. Conf. on Parallel Processing*, St. Charles, Illinois, 14-19.

[PP1] V. Y. Pan and F. P. Preparata. Supereffective slow-down of parallel algorithms. *Proc. 1992 ACM Symp. on Parallel Algorithms and Architectures*, San Diego, California, 402-409.

[PP2] V. Y. Pan and F. P. Preparata. Work-preserving speed-up of parallel matrix computations. *SIAM J. Comput.*, Vol. 24, No. 4, 1995.

[PR1] V. Y. Pan and J. H. Reif. Fast and efficient solution of path algebra problems. *Journal of Computer and System Sciences*, 38(3):494-510, June 1989.

[PR2] V. Y. Pan and J. H. Reif. The parallel computation of minimum cost paths in graphs by stream contraction. *Information Processing Letters* 40, 79-83, 1991.

[R] R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM J. Comput.*, Vol. 14, No. 2, 396-409(May 1985).

[S] R. Seidel. On the all-pair-shortest-path problem. *Proc. 24th ACM STOC*, ACM Press, 745-749(1992).

[T] T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Info. Process. Lett.* 43, 195-199(1992).

[V] L. G. Valiant. Parallelism in comparison problems. *SIAM J. Comp.*, Vol. 4, No. 3, 348-355(Sept. 1975).

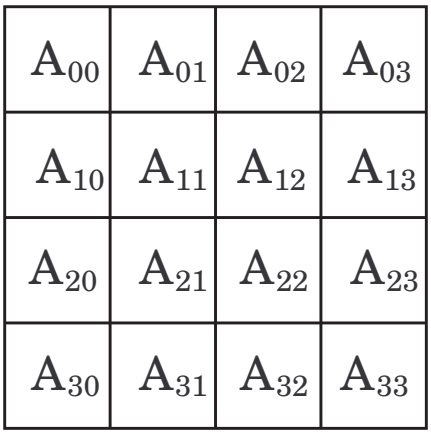
A_{00} A_{01} A_{02} A_{03}
A_{10} A_{11} A_{12} A_{13}
A_{20} A_{21} A_{22} A_{23}
A_{30} A_{31} A_{32} A_{33}

Fig. 1.

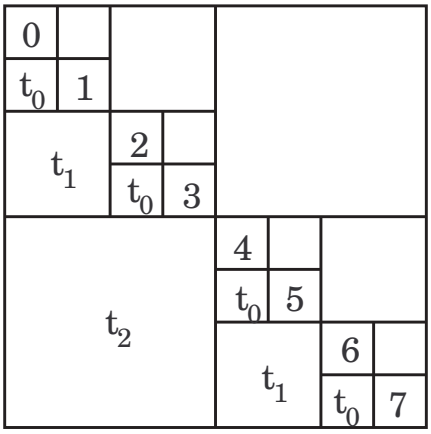
0
t_0 1
t_1 2
t_0 3
t_2 4
t_0 5
t_1 6
t_0 7

Fig. 2.